

道生能力接入开发手册.

目录

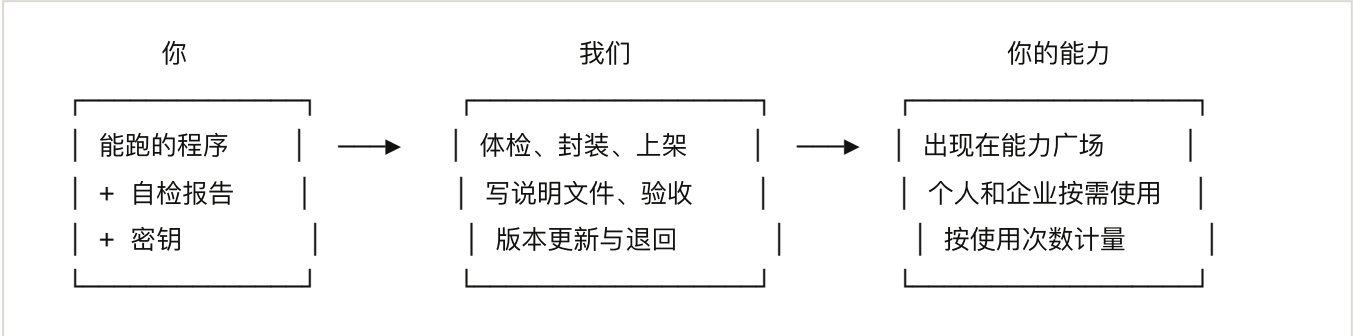
1. 第 0 章 · 一页看懂
2. 第 1 章 · 这是什么合作
3. 第 2 章 · 三种接入形式
4. 第 3 章 · 你的程序会在什么环境里跑
5. 第 4 章 · 三道硬门槛
6. 第 5 章 · 先自己检查一遍
7. 第 6 章 · 交付材料
8. 第 7 章 · 从提交到上架
9. 第 8 章 · 版本更新与退回
10. 第 9 章 · 自己做成品包（选读）
11. 第 10 章 · 常见问题
12. 附录 A · 一份通过的自检报告（节选）
13. 附录 B · 一个从头到尾的例子

版本	1.0 (2026-09)
给谁看	想把自己的程序、工具插件或方法接进道生的开发者和专家。技术负责人读第 0 章即可，动手的人读全篇
读完要多久	30 分钟
配套	同一个开发者包里的自检工具 <code>tools/intake-check.py</code> 、三套模板 <code>templates/</code> 、 <code>提交清单.md</code>

这一份就是全部。你不需要读我们的其他文档，也不需要了解道生的内部实现。读完这一份，你应当能独立把东西交对。

如果读完还有必须问我们才能回答的问题，那是这份手册的缺陷，请直接告诉我们。

第 0 章 · 一页看懂



三句话讲清分工

1. 你负责程序本身能用。我们不负责调试你的代码。
2. 我们负责把它接进道生。封装、说明文件、校验、上架、部署，都由我们做。
3. 发现问题我们告诉你，由你来修。我们只做格式转换，不改你的业务逻辑。一旦我们替你修，你每更新一次，我们就要重做一次，问题的主人也变成了我们。

只有 10 分钟的话，先跑这一条

```
python3 tools/intake-check.py \  
  --ok      "<你的程序正常跑通的命令>" \  
  --bad     "<参数错或缺依赖时的命令>" \  
  --verdict "<会得出坏结论的命令，比如查出了问题>"
```

它会告诉你哪几处会让我们卡住。把生成的报告一起提交，就是交付材料的一部分。

第 1 章 · 这是什么合作

1.1 道生是什么

道生是一个装在用户自己电脑或机房里的 AI 底座。用户用一句话说想做什么，道生从能力广场里挑合适的能力去做。你的能力接进来之后，就是能力广场里的一项，可以被个人和企业按需使用。

每个使用方一套独立的道生。 超级个体版装在个人电脑上，行业定制版装在企业机房或私有云里。使用方之间互相隔离，不共享进程，也不共享数据。这一点会影响你的密钥怎么交（见 6.2）。

1.2 谁做什么

事情	谁做
程序本身能跑、结果正确	你
跑一遍自检，把报告交来	你
提供可用的密钥，并说清归属	你
体检（授权、漏洞、隔离环境里试运行）	我们
封装（写适配层、写说明文件）	我们
校验、试运行、上架	我们
部署、权限、计量、监控	我们
出了问题判断是谁的问题	我们
修你代码里的缺陷	你（我们会告诉你是哪一条）

1.3 你会得到什么

- 你的能力出现在道生能力广场，有自己的详情页：写清能做什么、适合谁、体检结果、现在的状态。
 - 使用方按需使用，每次使用都有记录，按使用次数计量。收入分成的比例和结算方式以双方合同为准。
 - 每次上架或更新，我们给你一份《接入回执》：测出了什么、要你改什么、改完怎么办（样例见 7.4）。
-

第 2 章 · 三种接入形式

先判断你手上的东西属于哪一种。按顺序问，第一个回答「是」的就是它的形式。

问题	形式
调用一次就能给出确定格式的结果，中途不需要「先看看再决定」？	命令行程序（最常见）
你已经有一个工具插件服务（遵循 MCP 协议），想让道生调用其中的工具？	MCP 工具插件
它是一套做事方法——换个人做结果会不一样，靠的是经验和规则，不是一段程序？	技能

另外两种情况由我们来判断，你不用操心：

- 要跑很久、要花钱、要人批准、会产出文件的：我们会封成**作业**，跑完先出草稿，由人放行后才算数。
- 要一直盯着某个文件夹或数据源的：我们会封成**常驻任务**。

2.1 命令行程序

道生用一条命令调用你的程序，读取它打印出来的结果。你的程序用什么语言写都可以，只要能在命令行里运行。模板：`templates/example-forecast-1.0.0/`。

2.2 MCP 工具插件

如果你已经有一个 MCP 工具服务，不需要改写。我们用一层很薄的适配程序去启动你的服务、调用指定的工具、把结果交回道生。你需要保证：

- 工具服务能用一条命令在本机启动（通过标准输入输出通信）；
- 每个工具的名称、参数和返回内容稳定；
- 第 4 章的三道门槛同样适用：查出问题要正常返回结论，不要用报错来表达。

模板：`templates/example-text-check-1.0.0/`，里面附了一个能跑通的最小工具服务和适配程序。

2.3 技能

技能是一份写给 AI 看的做事方法：什么时候用、按什么步骤做、判断标准是什么、有哪些样例。它不是程序，所以不跑自检工具；我们用你提供的样例来验收。模板：`templates/example-weekly-report-1.0.0/`。

写技能时最要紧的两点：

1. 开头一句话写清「做什么 + 什么时候用 + 不做什么」。道生靠这句话决定要不要用它。

2. 规则写成清单，样例写成「输入 → 期望输出」。规则放在文件里，以后改规则不用改程序。

第 3 章 · 你的程序会在什么环境里跑

这一章请务必读。不知道这些，写出来的东西在道生里多半跑不起来，然后就要来回几轮。

3.1 我们怎么调用你

我们执行：	<你的命令>	动作	参数名=值	参数名=值
我们读取：	标准输出里的	JSON	← 结果	
	标准错误里的文字		← 给人看的日志（我们保留，用于排查）	
	退出码		← 跑没跑成	

- 参数用「参数名=值」的形式。如果你现在是 `--flag value`，不用改，我们在适配层里转换。但请不要只靠位置区分参数（`prog a b c`）：其中一个值为空时，后面的参数会整体错位，而且不会报错。
- 标准输出只打印 **JSON**。进度、警告、诊断信息一律打到标准错误。两者混在一起，我们就解析不了。

3.2 六条运行约束

#	约束	对你的程序意味着什么
1	多数能力写不进磁盘	只读的能力会在隔离环境里运行，除了指定目录，任何地方都写不进去，包括你程序自己的目录。要写文件，就写到环境变量 <code>DAOSHENG_TENANT_OUT</code> 指定的目录
2	密钥只从环境变量读	我们把密钥注入环境变量，变量名和你约定（如 <code>ACME_API_KEY</code> ）。不要从命令行参数读密钥：参数会进操作记录，等于把密钥写进日志
3	没有终端	程序在后台服务里运行。不能等键盘输入、不能弹窗、不能要人按 y 确认。等输入的表现是「卡住直到超时」，很难排查
4	可能被同时调用多次	一次处理一批目标是常态。共用的临时文件名、全局锁、写死的端口都会互相冲突
5	可能被重跑	失败重试、用户再点一次都会发生。如果重跑会产生重复的后果（重复发布、重复下单、重复计费），一定要告诉我们
6	能不能联网看部署	有的使用方能连外网，有的不能。你依赖哪些外部地址，请在材料里写清

3.3 超时与费用

- 我们会给每项能力设**超时**，这个数由自检报告里的实测耗时推算。正常就要跑好几分钟的，请说一声，我们会封成作业。
- 如果调用会**花钱**，请告诉我们怎么计费（按次、按量还是包月）。道生有用量上限，不知道计费方式的话，花的钱在账上看不见。

第 4 章 · 三道硬门槛

分三档，按它会被用在哪儿定。你只需要保证第一档的三条；第二档、第三档由我们实测，不用你申报。

第一档 · 能被调用（三道硬门槛）

#	要求	说明
1	能跑通	交之前自己把成功和失败两条路都跑一遍
2	返回结构稳定	字段可以是空值，但结构不能时有时无
3	「结果不合格」不等于「运行出错」	这是最要紧的一条，见下

第 2 条怎么理解

- 同一个动作的所有调用，用同一套结构，不管结论好坏。「查出问题」和「没查出问题」是同一个动作的两种结论，应当返回同一套结构：

```
# ✅ 一套结构，正常时列表为空
{"violations": [], "count": 0, "all_ok": True}
{"violations": [{...}], "count": 5, "all_ok": False}

# ❌ 两套结构，下游要写两份解析
{"items": [...], "count": 3}                # 正常时
{"violations": [...], "message": "5 处违规"} # 查出问题时
```

- 最外层的字段名必须固定，里面嵌套的字段名也应当固定。
- 没有值就给 `null` 或空列表，不要靠「不输出这个字段」来表达。
- 不同的动作之间不要求结构一致。

第 3 条展开说

查出了问题、没查到数据、判定为「否」，**这些都是成功的调用**，只是结论不好。请正常返回，把结论放在内容里；**不要用报错或非 0 退出码来表达。**

为什么这么要紧：道生把非 0 退出码一律当成「这项能力坏了」。于是**你查出的结论根本到不了用户手里**。我们真实吃过这个亏：一项体检能力连续两个月显示「没有发现问题」，其实它每天都在正常发现问题，只是结论被退出码吞掉了。

```
# ❌ 查出 5 处问题
sys.exit(1)

# ✅ 查出 5 处问题
print(json.dumps({"issues": [...], "all_ok": False}))
sys.exit(0)      # 一律 0，除非程序自己出错
```

退出码约定

退出码	含义
0	跑成了，结论好坏都在返回内容里
2	参数不对，或按规则拒绝执行
4	没找到、缺依赖、上游连不上
5	执行出错

第二档 · 能串进多步流程

多步流程会重跑、会夹在别的步骤中间，所以多四条。这些我们实测，你不用申报：

- 重跑不产生重复后果（做不到的话，我们标注「不可重跑」，流程里避开重跑）；
- 批量处理时，哪些成功、哪些失败，在返回里分得开；
- 大的产物写到文件里，返回文件路径，不要整个打到标准输出；
- 耗时有稳定的上限。

第三档 · 能直接给不懂技术的人用

输入来自不懂技术的人，所以再多四条。同样由我们实测：

- 输入有问题时，给出能自己改正的提示：缺列、空文件、编码乱、格式不对时，报「第 3 列缺少 date 字段」，而不是一大段程序报错。用户一定会给不完美的素材，这不是用户的错，是设计时就要考虑的前提；
- 调用前能估出大概多久、花多少；
- 能被同时调用多次；
- 不需要终端。

档位不是评价，是用途。第一档的能力同样有价值，只是不会直接放进给终端用户的界面里。

第 5 章 · 先自己检查一遍

5.1 跑自检工具

开发者包里的 `tools/intake-check.py` 是一个 Python 文件，不用安装任何东西，只用 Python 自带的标准库，也不看你的源码，只从外面观察程序的表现。

```
python3 tools/intake-check.py \
  --name      "行业需求预测" \
  --ok        "python3 forecast.py --symbol ACM-100" \
  --bad       "python3 forecast.py --symbol NOPE-999" \
  --verdict   "python3 forecast.py --symbol ACM-100 --strict" \
  --out       report.md
```

三条命令的含义，第三条最重要：

参数	给一条什么命令
<code>--ok</code>	正常跑通的一次调用
<code>--bad</code>	运行出错：参数错、缺依赖、上游连不上
<code>--verdict</code>	结果不合格：查出了问题、没查到数据、判定为否。这是成功的调用

它检查十一项，其中七项必须通过：标准输出是不是纯 JSON、日志有没有打到标准错误、两条路径的退出码、坏结论有没有被表达成运行出错、多次调用结构是否稳定、是否需要终端、命令行里有没有明文密钥；另外还会记录耗时分布和产物大小。

把生成的 `report.md` 一起提交。它同时代替了「输出样例」这项材料，所以你的总工作量是减少的。

5.2 对照表：最常让我们返工的五种写法

情形	❌ 会出事	✅ 这样写
查出问题	<code>sys.exit(1)</code>	<code>print(json.dumps({..., "all_ok": False})) + sys.exit(0)</code>
给人看的日志	<code>print("正在拉取...")</code>	<code>print("正在拉取...", file=sys.stderr)</code>
密钥	<code>--api-key sk-xxx</code>	<code>os.environ["ACME_API_KEY"]</code>
大产物	把 50MB 的 JSON 打到标准输出	写到文件，返回 <code>{"path": ".../out.json"}</code>
上游挂了	悄悄用缓存或假数据顶上	二选一，见下

最后一条最要紧。悄悄顶上会让调用方以为拿到的是真数据，比直接报错危险得多，也最难查。

上游挂了怎么办：两种做法都可以，但必须选一种并说清楚

	做法	退出码	返回内容	自检时放在
A (推荐)	直接失败	非 0 (上游可达用 4)	随意	<code>--bad</code>
B	用缓存照常返回	0	必须带 <code>"degraded": true</code> 和 <code>"degraded_reason": "..."</code>	<code>--ok</code>

选 B 时有两条硬要求：必须带 `degraded` 标记；不要用 `error` 字段表达降级——那会被读成「跑失败了却报成功」。

5.3 自检工具测不出来的，请你回答

程序从外面看不出业务含义。提交时直接回答这四个问题就行：

- 1. 重跑会不会产生重复后果？（重复发布、下单、计费）
- 2. 批量处理时部分失败，怎么表达？
- 3. 上游挂了会怎样？ 直接失败，还是用缓存顶上？
- 4. 能不能被同时调用多次？

第 6 章 · 交付材料

6.1 七样东西

#	交什么	说明
1	可运行的程序	一个目录、一个代码仓库链接、一个软件包名或一个容器镜像。必须能在 macOS 或 Linux 上运行；只能在其中一种系统上运行的，说一句
2	运行命令	一行命令，带全部参数。写在 README 里也行
3	输入说明	每个参数四件事：名字、类型、必填还是选填、一个真实的示例值
4	自检报告	第 5 章生成的 <code>report.md</code>
5	依赖清单	<code>requirements.txt</code> 、 <code>package.json</code> 、 <code>go.mod</code> 之类；没有的话，用一句话说明要装什么
6	可用的密钥	见 6.2
7	三句话定性	① 属于哪个领域（一个词）；② 会不会改动什么（只读查询 / 会写数据 / 会对外发布或下单）；③ 调用一次要不要花钱，怎么计费

没有格式要求。一个压缩包、一个仓库链接都可以。技能只需要第 1、3、7 项和样例。

第 7 项只要三句话，但不给的话我们只能猜。这三件事分别决定：它被放进哪个分类（影响用户能不能找到）、按多危险对待（影响要不要人批准）、花的钱能不能记到账上。

6.2 密钥

如果你的能力要调用你自己的服务或第三方接口，**请提供可用的密钥**，没有密钥我们跑不通，也就没法验收。

请一并说清四件事：怎么申请和更换、有没有调用次数或速度限制、是测试用还是正式用、**多个使用方时怎么办**。

最后一条最容易漏。道生是每个使用方一套，你的能力会装到很多使用方那里：

- **A. 共用你提供的这一把**：调用量和费用记在你的账上。每套道生都有固定的实例编号，我们可以按你的要求把它放进环境变量（如 `DAOSHENG_INSTANCE_ID`），由你的程序放进请求里，用来区分来源。**告诉我们你要什么格式**。
- **B. 每个使用方各自申请**：请提供申请指引，我们在安装时引导使用方自己申请。

请明确选 A 还是 B。不说清楚的话，每装一套新的道生都要回来问一次。

我们怎么保管：存进道生的密钥库，只在所在机器上，文件权限只有本机服务可读；不写进程序包、不进版本库、不出现在日志和返回内容里。你随时可以作废。

建议给一把专用的、可以随时作废、有额度限制的密钥，不要给你们的主密钥。

第 7 章 · 从提交到上架

7.1 四种来源都可以

来源	你给我们什么
代码仓库链接	公开或私有仓库都行（私有仓库请开只读权限）。我们固定到一个具体版本后再体检
软件包	已发布到常见软件包仓库的程序，给包名和版本号
容器镜像	镜像地址和启动命令，适合要一直运行的服务
专家方法文档	你的判断规则、表格模板和样例，我们帮你做成能力或技能

7.2 怎么提交

- **在线提交**：道生官网「能力广场 → 提交你的能力」，填能力名称、链接、它能做什么、联系方式。
- **加微信**：官网「加微信聊聊」，扫码后备注「接入能力」。
- **附件**（自检报告、依赖清单等）在我们回复后按约定方式发送。**密钥不要通过表单或聊天发送**，我们会单独约定安全的交付方式。

7.3 四个状态

状态	在做什么	你要做什么	大概多久
待评审	我们查来源、活跃度、授权和安全公告，出一份《接入评审单》	缺材料时补	两个工作日内给结论
体检中	固定版本、查授权、查漏洞、在隔离环境里试运行，写说明文件	发现问题时你来修	看复杂度
试用	以试用版上架，在真实任务里跑一段时间	不用	视使用情况
已上架	转为正式版，出现在能力广场	验收确认	—

评审结论只有三种：

结论 什么情况

接入 授权清楚、近期有维护、能在使用方自己的环境里运行、没有未修复的高危漏洞

观察 有价值，但授权需要进一步判断，或活跃度、安全还要再核

不接 没有授权声明、已停止维护、授权明确不允许道生的用法、或和现有能力完全重复

开源项目的授权，要分三种用法判断：

1. 道生自己内部使用；
2. 使用方自己安装，道生只提供连接；
3. 道生把它打包进产品、替使用方托管或转售。

很多带附加条件的授权只允许第 1 种。可见不等于可以随使用，**授权不清楚的，我们不会上架。**

7.4 你会收到的回执长这样

接入回执 · 行业需求预测

****结论****：接入，封成「命令行程序」（单次调用、结果确定、不需要人批准）

****我们测出来的****

- 耗时：3 次 1.2 ~ 2.8 秒 → 超时定为 30 秒
- 产物：约 12KB，直接返回，不用写文件
- 只读，不写磁盘 → 在隔离环境里运行
- 需要密钥：ACME_API_KEY（选 A，共用你方那把）

****要你改的（2 条）****

1.  `--strict` 查出问题时退出码为 1 — 请改成 0，结论放进返回内容。
现在这样写，你查出的问题到不了用户手里。
2.  进度日志打在标准输出里，和 JSON 混在一起 — 请改到标准错误。

****不用你改的****

- 参数是 `--flag value` 形式 — 我们在适配层里转换，你不用动

****改完怎么办****

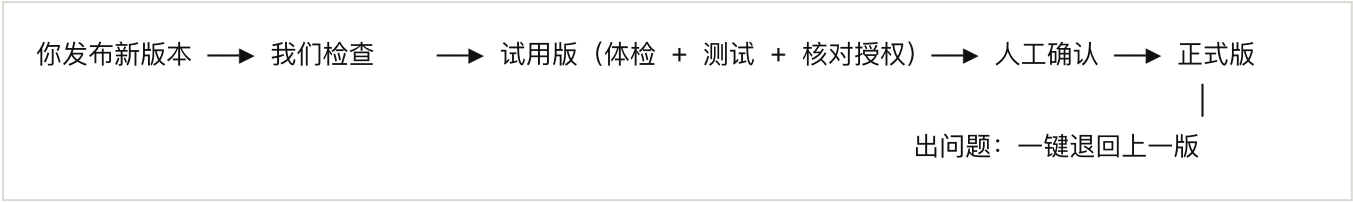
把改好的版本和新的自检报告发来，我们重新验收。

第 2 条不改也能上，但第 1 条必须改。

第 8 章 · 版本更新与退回

你的程序更新了，**请告诉我们**。我们会重新体检、重新验收，这一步不能省：一次看起来无害的改动，也可能让说明文件不再成立（多了一个字段、少了一个字段、退出码变了）。

更新走两个通道：



- 使用方按约定的时间更新到正式版。
- 超过 180 天没有更新、授权发生变化、出现高危漏洞的，会降级或下架，并提前通知你。

版本号怎么递增

改动	版本号
修正、优化，不影响调用方式	递增第三位或第二位，如 1.2.0 → 1.2.1
破坏性改动：删除或改名参数、收紧参数取值、改变返回结构、只读变成会写、需要更高权限	递增第一位，如 1.2.0 → 2.0.0；旧版本保留一段过渡期

第 9 章 · 自己做成品包（选读）

你不需要做这一步，说明文件和目录结构都由我们来写。但如果你想自己交一个可以直接安装的成品包，照着模板做即可。

9.1 目录结构

你的能力-1.0.0/

— capability.yaml	说明文件（必需）
— run.py	适配程序（命令行程序和 MCP 工具插件需要）
— README.md	做什么、需要哪些密钥、依赖哪些外部服务、联系人
└─ examples/	
└─ smoke.json	两三个调用样例，同时用于试运行

包内引用自己的文件时，一律写成 `{PKG}/文件名`，安装时会换成实际路径。

9.2 说明文件逐项解释

```

spec_version: "1.0"

package:
  id: example.forecast          # 在道生里的唯一名字，由我们确认
  version: "1.2.0"
  vendor: "示例公司"
  contact: "填你的联系方式"
  license: "proprietary"       # 开源项目写授权名，如 MIT、Apache-2.0

runtime:
  os: [darwin, linux]         # 能在哪些系统上运行；安装时据此拒装
  python: ">=3.9"
  needs_network: wan          # none 不联网 / lan 只连内网 / wan 连外网
  needs_secrets: [ACME_API_KEY] # 需要注入的密钥变量名

capability:
  tag: ""                      # 分类，由我们按领域词表填写，可以留空
  kind: cli
  desc: "取行业需求预测。做产能规划或备货判断时用；只出预测不出建议。"
  entrypoint: "python3 {PKG}/run.py forecast symbol={symbol} horizon={horizon}"
  params:
    symbol: { type: string, required: true, desc: "标的代码", examples: ["ACM-100"] }
    horizon: { type: integer, default: 30, minimum: 1, maximum: 365, desc: "预测天数" }
  returns:
    format: json
    max_bytes: 200000
    schema: "{symbol, horizon, points:[{date, value}], model_version}"
  grain: slow                  # quick 5 秒内 / slow 5 分钟内 / batch 更久
  timeout_s: 120
  net: wan
  cost_class: zero             # zero 不花钱 / paid 花钱 / mixed 部分花钱
  risk_level: GREEN            # GREEN / YELLOW / RED
  isolation: readonly          # readonly 只读 / readwrite 会写数据
  attributes: { readOnly: true, destructive: false, idempotent: true, openWorld: true }
  lifecycle: experimental      # experimental 试用 / production 正式 / deprecated 停用
  secrets: [ACME_API_KEY]
  health: "python3 {PKG}/run.py --health"
  examples:
    - args: { symbol: "ACM-100" }

```

最要紧的五个字段：

字段	为什么要紧
desc	它决定道生在什么时候会想到用你。写清「做什么 + 什么时候用」，最好再写一句「不做什么」。不超过 1024 个字符
params 里的 examples	给出真实的示例值，调用准确率明显更高；这比反复打磨 desc 的措辞更有效
entrypoint 与 params 对应	params 里的每个名字都要在 entrypoint 里以 {名字} 出现，反过来也一样。对不上的会被拒绝安装
选填参数的 default	每个选填参数都要给默认值。没给默认值又没传的，命令里会少一个值，而且不会报错
attributes	会不会改数据、能不能重跑。没写的一律按最危险的情况对待，会要求人批准

要人批准的动作（对外发布、下单、群发、动钱）：在 capability 里加一行 gate: human_approve，道生会拦下自动调用，必须由人放行。

9.3 三套模板

模板	适合	怎么用
templates/example-forecast-1.0.0/	命令行程序	复制目录，改 capability.yaml 和 run.py 里的 forecast 逻辑
templates/example-text-check-1.0.0/	已有 MCP 工具服务	把 mcp_server_example.py 换成你的服务启动命令，在 capability.yaml 里写清工具名
templates/example-weekly-report-1.0.0/	做事方法	改 SKILL.md 的开头说明、步骤、判断标准和样例

每套模板都能直接跑通自检（技能模板除外，技能用样例验收）。先跑通模板，再换成你自己的逻辑，最省事。

第 10 章 · 常见问题

问：我的能力是一个网络服务，不是命令行程序，可以吗？ 可以。第 2 项写成「服务怎么启动 + 调用哪个地址 + 请求内容是什么」即可。自检工具也能测：把 `--ok` 写成一条 `curl` 命令就行。

问：我的程序依赖内网资源、特定硬件或只有我们有的数据库，可以吗？ 可以接，但要在依赖清单里写清楚。这类能力只能装在能访问到这些资源的使用方那里，我们会在详情页上注明。

问：跑一次要很久、要花钱、要人确认，可以吗？ 可以，但它可能会被封装成「作业」而不是一次调用：跑完先出草稿，由人放行后才算数。这个判断由我们来做。

问：程序还有已知问题，可以先交吗？ 请先修。我们不负责调试。封装过程中发现的问题我们会告诉你，由你来修。

问：一定要按你们的格式打包吗？ 不需要。说明文件、目录结构、校验都由我们来做。想自己交成品包的，看第 9 章。

问：交完多久能上架？ 取决于复杂度和依赖。收到材料后两个工作日内，我们先给评审结论和大致时间。

问：我怎么知道自己交得对不对？ 跑自检工具。必须通过的七项都通过，就可以交了。

问：收入怎么分？ 按使用次数或按月计量，分成比例、结算周期和最低门槛以双方合同为准。每一笔分成都对应可以核对的使用记录。

附录 A · 一份通过的自检报告（节选）

```
# 道生 · 能力接入自检报告

- 能力：行业需求预测
- 结论：必过项全部通过

| | 检查项 | 是否必过 |
|---|---|---|
|  | 命令能跑起来 | 必过 |
|  | 标准输出是不是纯 JSON | 必过 |
|  | 成功路径退出码 | 必过 |
|  | 失败路径退出码 | 建议 |
|  | 坏结论 ≠ 运行出错 | 必过 |
|  | 返回结构是否稳定 | 必过 |
|  | 是否需要终端 | 必过 |
|  | 密钥有没有走命令行 | 必过 |

###  坏结论 ≠ 运行出错
退出码 0，结论在返回内容里。这正是我们要的写法。

###  耗时
3 次：0.3 ~ 1.5 秒。我们据此设超时，你不用猜。
```

附录 B · 一个从头到尾的例子

你交了什么：一个目录，里面有 `forecast.py`，运行起来是这样：

```
$ python3 forecast.py --symbol ACM-100
{"symbol": "ACM-100", "points": [{"date": "2026-09-05", "value": 101}], "model_versi
```

再加上自检报告、`requirements.txt`、一把测试用的密钥，以及三句话：「行业需求预测；只读查询；按次计费」。

我们做了什么

1. 评审：单次调用、结果确定、不需要人批准 → 封成命令程序；
2. 实测：耗时 1.2 ~ 2.8 秒 → 超时定 30 秒；只读不写盘 → 在隔离环境里运行；产物 12KB → 直接返回；
3. 写说明文件：其中那句 `desc` 决定了道生在什么时候会想到用你的能力，你交的三句话主要就用在 这里；
4. 写适配层：把 `--symbol X` 转成「参数名=值」的形式，把密钥从环境变量里取出来。你的代码一行没动；
5. 试用、上架。

用户最终怎么用它：用户不知道有 `forecast.py`，就是说一句话：

用户：帮我看看 ACM-100 未来三个月的需求走势，做个备货判断

道生：（判断需要预测数据 → 调用你的能力 → 拿到结果 → 结合上下文回答）未来 90 天 ACM-100 的需求预测呈现.....建议备货区间在.....

如果你的能力查不到这个标的，正确的表现是：

道生：ACM-100 在库里查不到。要不要换个代码，或者按同行业的平均水平估算？

——道生知道发生了什么，因为你把「查不到」当成结论返回了。如果你用退出码 1 表达「查不到」，用户看到的只会是：

道生：这项能力调用失败了。

这就是第 4 章那三道硬门槛存在的全部理由。

本手册如与我们内部的技术规范冲突，由我们负责协调，你以本手册为准。
